GiGA IoTMakers

C SDK 매뉴얼_TCP

Document info

Document name: "C SDK 매뉴얼_TCP"

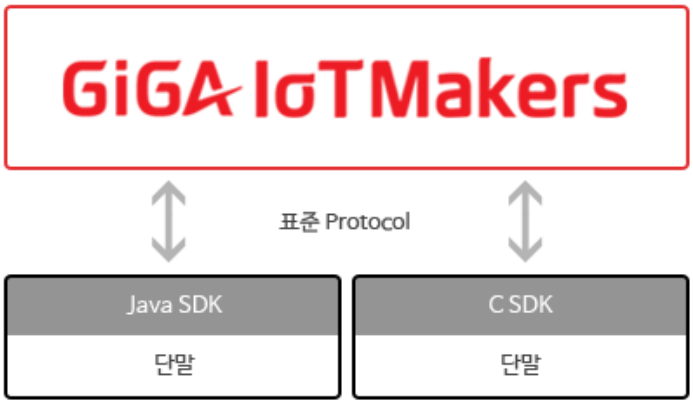
Document date: 2016-04-06

Manual release version: 2.2.0

목 차

1 개요

IoTMakers SDK(Software Development Kit)는 IoT 디바이스(이하 디바이스)가 IoTMakers 플랫폼에 연동할 때에 필요한 API(Application Programming Interface)를 제공한다. 디바이스는 제공되는 API 를 통하여 접속 및 장비인증, 수집데이터 전송하고, 플랫폼으로부터 제어데이터를 수신한다.



[그림 1] IoTMakers SDK 연동 구조

디바이스가 IoTMakers 플랫폼에 연동하기 위해서는 IoTMakers 포털에서 디바이스 등록과정을 거쳐야 한다. 자세한 내용은 IoTMakers 포털(http://iotmakers.kt.com)에서 확인한다.

이 문서는 IoTMakers SDK 를 사용하는 프로그래머에게 프로그래밍 가이드와 레퍼런스를 제공한다.

2 C SDK 소개

C SDK 는 Unix/Linux 기반의 C/C++언어 환경에서 사용할 IoTMakers 연동용 C API 를 제공한다. 이 SDK 는 소스 형태로 제공되므로 각 개발 환경에서 라이브러리 형태로 빌드하여 어플리케이션에 개발에 사용한다. 표준 C 소스코드를 컴파일 할 수 있는 컴파일러와 ptrhead 를 제공하는 환경이 요구된다.

2.1 SDK 다운로드

IoTMakers 포털에 로그인 후에 해당 SDK 를 내려 받는다. 내려 받은 SDK 를 원하는 작업폴더에 저장 후에 압축을 푼다.

```
$ tar xvfz SDK_C_2.0.0_TCP.tgz
```

2.2 SDK 폴더 구조

```
$ cd ./SDK_C_2.0.0_TCP

$ ls -al
drwxrwxr-x  7 user user 4096  1월 15 16:39 .
drwxr-xr-x 39 user user 4096  1월 15 17:20 ..
drwxrwxr-x  2 user user 4096  1월 15 16:39 include
drwxrwxr-x  2 user user 4096  1월 15 16:39 lib
drwxrwxr-x  8 user user 4096  1월 15 16:40 src
drwxrwxr-x  2 user user 4096  1월 15 16:39 test
```

SDK 라이브러리의 소스코드는 src/ 폴더 안에 구성되어 있다. ./src/폴더 안의 주요 내용은 다음과 같다.

파일 및 폴더명	내용
base/	IoTMakers클라이언트 인스턴스 구현체
packet/	IoTMakers메시지 패킷 구현체이며, Header와 Body를 정의하고 조작. 장지인증, 수집데이터, 제어데이터 송수신에 사용되는 Body조작 구현
action/	IoTMaker연동 플로우 정의 구현체 장지인증, 수집데이터 전송, 제어데이터 수신 플로우를 구현
socket/	TCP/IP 소켓 인터페이스 구현체
parson/	JSON 파서 라이브러리
util/	log등의 유틸리티성 기능 구현체
im_common.h	API에 공통으로 사용되는 타입과 정의
iotmakers.h	개발자에게 오픈되는 API의 프로토타입 정의
Iotmakers_api.c	개발자 API의 구현체
Makefile	SDK라이브러리를 생성하는 Make스크립트

2.3 SDK 빌드 및 설치

C 프로그램 빌드환경 확인은 다음과 같이 한다.

```
$ which cc
/usr/bin/cc

$ which make
/usr/bin/make
```

cc(c compiler)와 make 가 설치되어 있지 않다면, 인터넷에 연결된 상태에서 다음 명령으로 설치한다.

```
#데비안 계열(ubuntu 등)에서

$ sudo apt-get install build-essential

# 또는
# RedHat 계열(CentOS 등)에서

$ sudo yum group install "Development Tools"
```

SDK 의 소스 폴더(./src/)로 이동 한 후에 빌드(make)를 진행하여 라이브러리를 생성한다. 빌드가 성공하면 ./lib 폴더 안에 ibiotmakers.a 파일이 생성된다

```
$ cd ./src

$ ls -al
-rwxrwxr-x 1 user user  1924  1월 15 16:39 Makefile
drwxrwxr-x 5 user user  4096  1월 15 17:34 action
drwxrwxr-x 2 user user  4096  1월 15 17:34 base
-rwxrwxr-x 1 user user  3416  1월 15 16:39 im_common.h
-rwxrwxr-x 1 user user 11935  1월 15 16:39 iotmakers_api.c
-rwxrwxr-x 1 user user   745  1월 15 16:39 iotmakers.h
drwxrwxr-x 2 user user  4096  1월 15 17:34 packet
drwxrwxr-x 2 user user  4096  1월 15 17:34 parson
drwxrwxr-x 2 user user  4096  1월 15 17:34 socket
drwxrwxr-x 2 user user  4096  1월 15 17:34 util

$ make clean

$ make
cc -W -O0 -g -D_LINUX_ -I./ -c parson/parson.c -o parson/parson.o
cc -W -O0 -g -D_LINUX_ -I./ -c socket/sock.c -o socket/sock.o
cc -W -O0 -g -D_LINUX_ -I./ -c util/log.c -o util/log.o
… [생략]
cc -W -O0 -g -D_LINUX_ -I./ -c base/base.c -o base/base.o
cc -W -O0 -g -D_LINUX_ -I./ -c iotmakers.c -o iotmakers.o
```

```
if [ -f ../lib/libiotmakers.a ]; then
/bin/rm ../lib/libiotmakers.a; fi;
ar
rvs ../lib/libiotmakers.a ./parson/parson.o ./socket/sock.o ./uti
l/log.o ./util/util.o ./packet/head_ext.o ./packet/head.o ./packe
t/body.o ./packet/packet.o ./packet/body_200.o ./packet/body_400.
o ./packet/body_500.o ./action/if200_auth/if224_auth_channel_dev_
tcp.o ./action/if200_auth/if231_auth_keepalive.o ./action/if400_c
ollection/if411_collection_data.o ./action/if500_control/if525_co
ntrol_data.o ./action/recv_packet.o ./action/send_packet.o ./base
/base.o ./iotmakers_api.o
ar: creating ../lib/libiotmakers.a


$ cd ..

$ ls -al ./include
-rwxrwxr-x 1 user user 1510  1월 15 16:39 iotmakers.h

$ ls -al ./lib
-rw-rw-r-- 1 user user 268242  1월 15 17:29 libiotmakers.a
```

SDK 빌드 후에 ./include/iotmakers.h 와 ./lib/ibiotmakers.a 이 있는지 확인한다.

2.4 어플리케이션 샘플소스 빌드

```
$ cd test
$ make clean
$ make
cc -W -O0 -g -I../include -c main.c -o main.o
cc -o tt main.o   ../lib/libiotmakers.a -lpthread
```

샘플 빌드 후에 실행파일(tt)이 생성되면 SDK 빌드 및 설치는 제대로 된 것이다. 실제 실행하고 IoTMakers 에 연동하기 위하여 정상 등록된 디바이스의 정보가 필요하다.

3 C API 목록

주요 API 는 다음과 같다. 모든 API 에는 접두어(im_)가 붙는다.

분류	API명	내용
IoTMakers 인스턴스	im_init()	IoIMakers연동을 위한 인스턴스 준비
	im_init_with_config_file()	IoIMakers연동정보를 파일로부터 읽어옴
	im_release()	인스턴스 자원해제
	im_get_LastErrCode()	가장 최근에 설정된 SDK오류코드 확인
	long long im_get_new_trxid()	수집데이터 전송시에 사용되는 트랜잭션 아이디를 생성
	void im_set_loglevel()	SDK의 로그레벨 지정
사용자 콜백	im_set_numdata_handler()	장비제어데이터 핸들러 등록(숫자타입)
	im_set_strdata_handler()	장비제어데이터 핸들러 등록(문자열타입)
	im_set_dataresp_handler()	수집데이터 전송결과 처리 핸들러 등록
	im_set_error_handler()	오류 처리 핸들러 등록
서비스	im_start_service()	IoIMakers연동 서비스 시작
	im_stop_service()	IoIMakers연동 서비스 종료
데이터 전송	im_send_numdata()	태그스트림이름과 숫자값을 전송
	im_send_strdata()	태그스트림이름과 문자열값을 전송
복합 데이터 전송	im_init_complexdata()	복합데이터 구조체 준비
	im_add_numdata_to_complexdata()	태그스트림과 숫자값을 추가
	im_add_strdata_to_complexdata()	태그스트림과 문자열값을 추가
	im_send_complexdata()	준비된 복합데이터 전송

참고로, IoTMakers 플렛폼에서 보내는 디바이스 제어데이터를 처리하기 위한 사용자 콜백(callback)함수의 타입은 다음과 같다.

구분	형	설명	비고
콜백	typedef void (*IMCbTagidNumDataHndl)(char *tagid, double val)	제어:숫자형 태그 데이터 처리 함수 타입	
	typedef void (*IMCbTagidStrDataHndl)(char *tagid, char *val)	제어:문자형 태그데이터 처리 함수 타입	
	typedef void (*IMCbDataRespHndl)(unsigned long long trxid, char *respCode)	수집:수집데이터 전송결과 처리 함수 타입	

	typedef void (*IMCbErrorHndl)(int errCode)	오류처리:오류코드 처리 함수 타입	

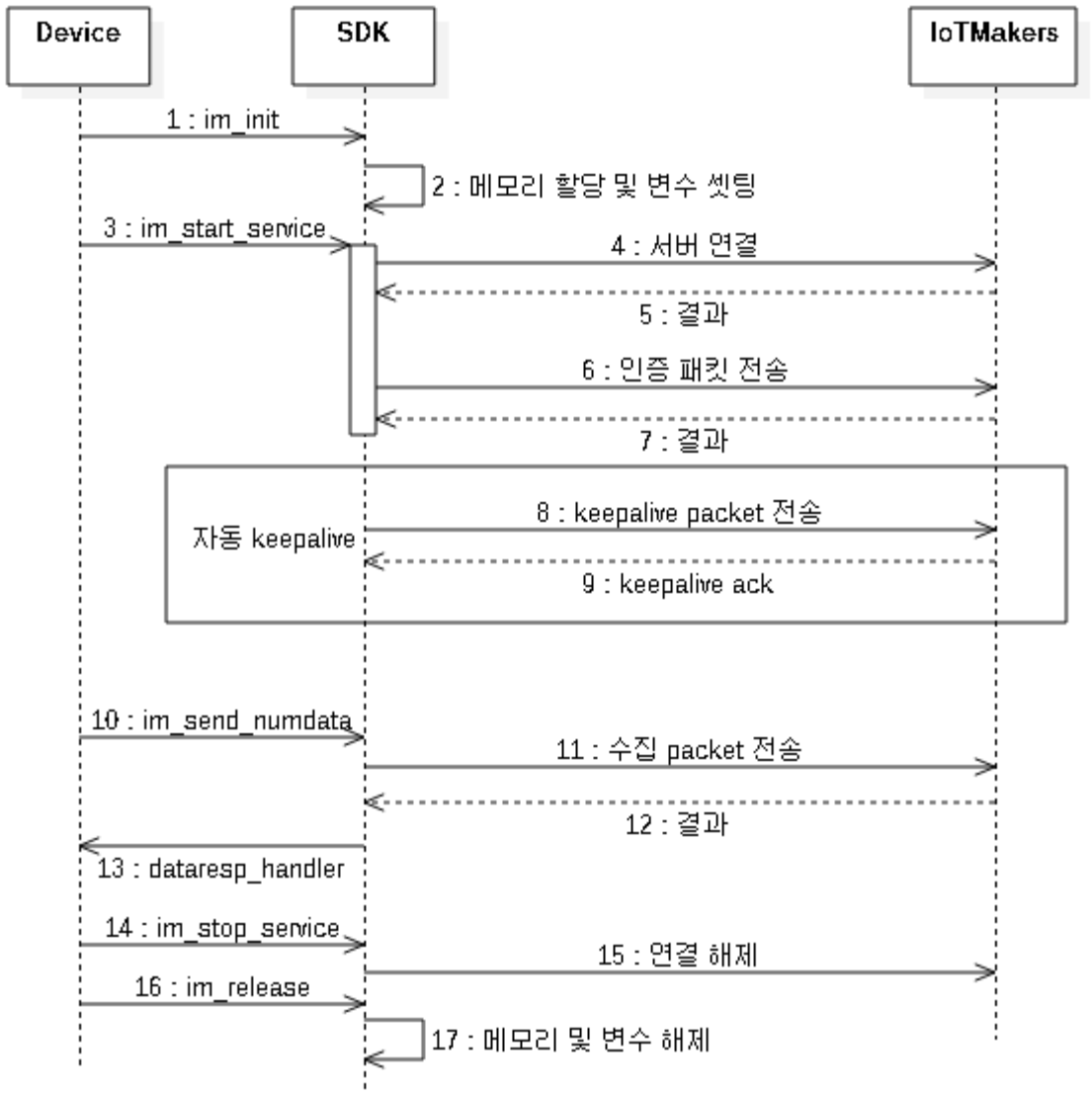
4 디바이스 연동(TCP 방식)

디바이스 연동을 하기 위하여 a) 디바이스 아이디, b) 디바이스 패스워드, 그리고 c) Gateway 연결 ID 가 필요하다. 이 정보는 IoTMakers 포털의 사용자 계정에서 새 디바이스를 등록한 후에 확인할 수 있다.

4.1 수집데이터 전송 시나리오

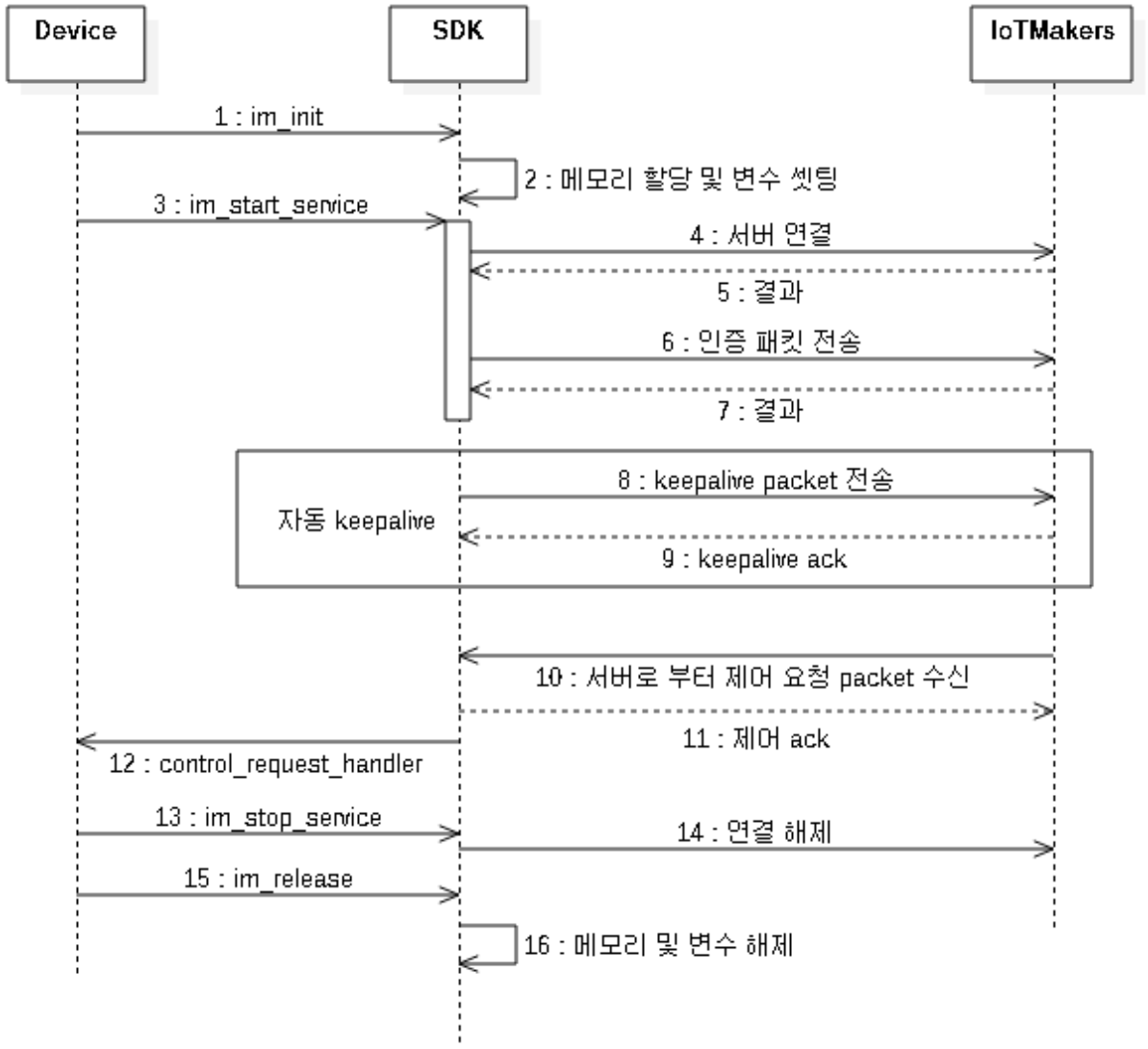
TCP Connection 연결후에, IoTMakers 플렛폼에 디바이스 채널의 인증을 시도한다. 인증시 디바이스아이디, 디바이스패스워드, Gateway 연결 Id 를 넘겨준다. 인증 성공시에는 현재 채널에 대한 인증번호(authNum)를 받는다.

디바이스에서 수집된 데이터를 IoTMaker 에 전송한다. 태그스트림아이디(tagId)와 수집데이터(val) 쌍을 전송한다. 데이터타입은 숫자형과 문자형을 전송할 수 있다. 또한, 단 건 전송 뿐만 아니라, 2 건 이상의 데이터를 동시에 전송할 수 있다.



4.2 제어데이터 수신 시나리오

IoTMakers 플렛폼에서 제어데이터를 디바이스로 전송하는 경우이다. 디바이스가 제어데이터 수신하면 즉시 응답하고, 장비제어 콜백를 통하여 제어처리한다. C SDK 는 내부적으로 looper 쓰레드 안에서 제어데이터를 수신하고, 데이터를 추출하여 사용자가 정의한 콜백 함수를 호출한다. 가능하다.



5 C API 상세설명

IoTMakers SDK 에서 제공하는 API 를 설명한다.

5.1 im_init()

IoTMakers 클라이언트 인스턴스를 초기화한다..

int im_init(char *ip, int port, char *deviceId, char *devicePasswd, char *gatewayId, char *logFilename)		
param	ip	IoTMakers서버 아이피
	port	IoTMakers서버 포트
	deviceId	디바이스 아이디
	devicePasswd	디바이스 패스워드
	gatewayId	등록된 디바이스의 Gateway연결ID
	logFilename	SDK에서 발생하는 로그기록파일의 이름
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	char *imServer = "220.90.216.90";
	int   imPort = 10020;
	char *devId = "xxxxxxx1450836201263";
	char *devPasswd = "xxxxxp55o";
	char *gwId = "OPEN_TCP_001PTL001_1999999999";
	char *logFile = "/tmp/iotmakers_sdk.log";

	rc = im_init(imServer, imPort, devId, devPasswd, gwId,
logFile);
	if ( rc < 0  )     {
		printf("fail im_init()\n");
		return -1;
	}

	// 생략
}
```

5.2 im_init_with_config_file()

설정파일의 디바이스 정보를 기준으로 IoTMakers 클라이언트 인스턴스를 초기화한다..

int im_init_with_config_file(char *fname)		
param	fname	설정파일의 값을 기반으로 인스턴스를 초기화 한다. (설정파일의 예시는 아래를 참조)
return	성공/실패	= 0: 성공 < 0: 실패

설정파일예시)

```
SERVER_IP=220.90.216.90
SERVER_PORT=10020
DEV_ID=XXXXXXD1450836201263
DEV_PW=XXXXXX55o
GATEWAY_ID=OPEN_TCP_001PTL001_1000999999
LOG_FILE_NAME=/tmp/iotmakers_sdk.log
```

코드예시)

```
#include <stdio.h>
#include <stdlib.h>

#include "iotmakers.h"

int main()
{
	char *configFile = "./config.txt";

	rc = im_init_with_config_file(configFile);
	if ( rc < 0 )	{
		printf("fail im_init_with_config_file()\n");
		return -1;
	}

	// 생략
}
```

5.3 im_release()

IoTMakers 클라이언트 인스턴스의 자원을 해제한다..

void im_release()		
param	N/A	
return	N/A	

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{

	// 생략

	im_release();

	return 0;
}
```

5.4 im_set_loglevel()

SDK 내부 로깅 레벨을 결정한다. 로그레벨은 3 개(ERROR, INFO, DEBUG) 중에 하나를 사용할 수 있다. 지정하지 않는 경우에는 ERROR 로 지정된다. Im_init()이후에 사용한다.

void im_set_loglevel(int loglevel)		
param	loglevel	Loglevel은 iotmakers.h에 정의되어 있다. #define LOG_LEVEL_ERROR 1 #define LOG_LEVEL_INFO 2 #define LOG_LEVEL_DEBUG 3
return	N/A	

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	// 생략
	im_set_loglevel(LOG_LEVEL_DEBUG);

	// 생략
}
```

5.5 im_get_LastErrCode()

API 사용 도중에 가장 최근에 설정된 SDK 내부 오류코드를 리턴한다.

int im_get_LastErrCode()		
param	N/A	
return	오류코드	sdk내부 오류코드 코드 리스트는 부록을 참조.

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	// 생략
	int sdkErrCode;
	sdkErrCode = im_get_LastErrCode();
	printf("sdkErrCode=[%d]\n", sdkErrCode);

	// 생략
}
```

5.6 im_get_new_trxid()

수집데이터 전송시에 사용하는 트랜잭션아이디를 생성한다. 사용예는 6 장의 샘플코드를 참고.

long long im_get_new_trxid()		
param	N/A	
return	trxid	트랜잭션아이디

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	// 생략
	long long trxid;
	trxid = im_get_new_trxid();
	printf("trxid=[%d]\n", trxid);
	// 생략
}
```

5.7 im_set_numdata_handler()

IoTMakers 에서 전송한 디바이스 제어데이터(숫자형)를 처리하기 위한 콜백함수를 등록한다.

void im_set_numdata_handler(IMCbTagidNumDataHndl cb_proc)		
param	cb_proc	사용자 정의 콜백함수(아래 코드예시 참조) 콜백함수에는 통상 디바이스 제어를 위한 코드를 작성
return	N/A	

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

static void mycb_numdata_handler(char *tagid, double val)
{
	printf("MY tagid=[%s], val=[%f]\n", tagid, val);
}

int main()
{
	// 생략
	im_set_numdata_handler(mycb_numdata_handler);
	// 생략
}
```

5.8 im_set_strdata_handler()

IoTMakers 에서 전송한 디바이스 제어데이터(숫자형)를 처리하기 위한 콜백함수를 등록한다.

void im_set_strdata_handler(IMCbTagidNumDataHndl cb_proc)		
param	cb_proc	사용자 정의 콜백함수(아래 코드예시 참조) 콜백함수에는 통상 디바이스 제어를 위한 코드를 작성
return	N/A	

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

static void mycb_strdata_handler(char *tagid, char *val)
{
	printf("MY tagid=[%s], val=[%s]\n", tagid, val);
}

int main()
{
	// 생략
	im_set_strdata_handler(mycb_strdata_handler);
	// 생략
}
```

5.9 im_set_dataresp_handler()

수집데이터 전송 후 응답 결과를 처리하기 위한 콜백함수를 등록한다.

void im_set_dataresp_handler(IMCbDataRespHndl cb_proc)		
param	cb_proc	사용자 정의 콜백함수(아래 코드예시 참조) 콜백함수 매개변수에는 트랜잭션아이디와 응답코드가 온다.
return	N/A	

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

static void mycb_dataresp_handler(long long trxid, char *respCode)
{
	printf("trxid=[%lld], respCode=[%s]\n", trxid, respCode);
}

int main()
{
	// 생략
	im_set_dataresp_handler(mycb_dataresp_handler);
	// 생략
}
```

5.10 im_set_error_handler()

수집데이터 전송 후 응답 결과를 처리하기 위한 콜백함수를 등록한다. 7.1 의 오류코드 표를 참조.

void im_set_error_handler(IMCbErrorHndl cb_proc)		
param	cb_proc	사용자 정의 콜백함수(아래 코드예시 참조) 콜백함수 매개변수에는 SDK내부 오류코드가 온다.
return	N/A	

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

static void mycb_error_handler(int lastErrCode)
{
	printf("lastErrCode=[%d]\n", lastErrCode);
	switch ( lastErrCode )
	{
	case 401:	//IM_ErrCode_SOCK_SEND_FAIL
	case 402:	//IM_ErrCode_SOCK_RECV_FAIL
	case 403:	//IM_ErrCode_SOCK_CONNECTION_FAIL
	case 404:	//IM_ErrCode_SOCK_CONNECTION_LOSS
		break;
	}
}

int main()
{
	// 생략
	im_set_error_handler(mycb_error_handler);
	// 생략
}
```

5.11 int im_start_service()

IoTMakers 클라이언트 서비스를 시작한다.

IoTMakers 플렛폼에 디바이스 채널의 인증을 시도한다. 인증시 디바이스아이디(devId), 디바이스패스워드(devPasswd), Gateway 연결 Id(gwId)를 넘겨준다. 인증 성공시에는 현재 채널에 대한 인증번호(authNum)를 받는다. 이 인증번호는 이후 데이터 교환시에 이용한다(MQTT, HTTP 사용시에 해당).

SDK 내부적으로 IoTMakers 서버에 접속, 디바이스 인증 후에, looper 를 가진 쓰레드를 생성한다. 서비스 시작 후에는 제어가 다시 사용자 프로그램 측으로 넘어온다.

인증 성공시에는 연결유지를 위하여 Keepalive 메시지를 주기척으로 전송하기 위하여 50 초 주기의 타이머가 설정된다.

int im_start_service()		
param	N/A	
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	int rc;

	// 생략
	rc = im_start_service();
	if ( rc < 0 )	{
		printf("lastErrCode[%d]\n", im_get_LastErrCode());
		return -1;
	}
	// 생략
}
```

5.12im_stop_service()

IoTMakers 연결을 종료하고, 클라이언트 서비스를 종료한다.

void im_stop_service()		
param	N/A	
return	N/A	

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	// 생략

	im_stop_service();

	// 생략
}
```

5.13 im_send_numdata()

숫자형의 수집데이터를 태그스트림의 이름으로 전송한다. 태그스트림은 IoTMakers 포털에 등록된 디바이스의 상세정보란에서 정의한다.

int im_send_numdata(char *tagid, double val, long long trxid)		
param	tagid	해당 디바이스에 등록된 태그스트림이름
	val	태그스트림 값 반드시 double형을 사용해야 한다.
	trxid	이 데이터에 대한 트랜잭션아이디를 명시한다. 0값을 사용하면, sdk가 자동생성한다.
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	int rc;
	// 생략

	char *tagID = "temperature";
	double numVal = 36.5;

	rc = im_send_numdata(tagID, numVal, 0);
	if ( rc < 0 )	{
		printf("lastErrCode[%d]\n", im_get_LastErrCode());
		return -1;
	}
	// 생략
}
```

5.14 im_send_strdata()

문자형의 수집데이터를 태그스트림의 이름으로 전송한다. 태그스트림은 IoTMakers 포털에 등록된 디바이스의 상세정보란에서 정의한다.

int im_send_strdata(char *tagid, char *val, long long trxid)		
param	tagid	해당 디바이스에 등록된 태그스트림이름
	val	태그스트림 값 반드시 '\0'으로 끝나는 문자배열을 사용해야 한다.
	trxid	이 데이터에 대한 트랜잭션아이디를 명시한다. 0값을 사용하면, sdk가 자동생성한다.
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	int rc;
	// 생략

	char *tagID = "switch";
	char *strVal = "OFF";

	rc = im_send_strdata(tagID, strVal, 0);
	if ( rc < 0 )	{
		printf("lastErrCode[%d]\n", im_get_LastErrCode());
		return;
	}
	// 생략
}
```

5.15 im_init_complexdata()

2 개 이상의 수집데이터를 동시에 전송하려면 복합데이터(complexdata)를 구성해야 한다. 복합데이터를 초기화하면 SDK 내부에 복합데이터 구조체가 초기화된다.

int im_init_complexdata()		
param	N/A	
return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	int rc;
	// 생략

	rc = im_init_complexdata();
	if ( rc < 0 )	{
		printf("lastErrCode[%d]\n", im_get_LastErrCode());
		return -1;
	}
	// 생략
}
```

5.16 im_add_numdata_to_complexdata()

숫자형의 수집데이터를 복합데이터에 추가한다.

int im_add_numdata_to_complexdata(char *tagid, double val)		
Param	tagid	해당 디바이스에 등록된 태그스트림이름
	val	태그스트림 값 반드시 double형을 사용해야 한다.
Return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	int rc;
	// 생략

	rc = im_add_complex_data_number("timer", (double)60);
	if ( rc < 0 )	{
		printf("lastErrCode[%d]\n", im_get_LastErrCode());
		return -1;
	}
	// 생략
}
```

5.17 im_add_strdata_to_complexdata ()

문자형의 수집데이터를 복합데이터에 추가한다.

int im_add_strdata_to_complexdata(char *tagid, char *val)		
Param	tagid	해당 디바이스에 등록된 태그스트림이름
	val	태그스트림 값 반드시 '\0'으로 끝나는 문자배열을 사용해야 한다.
Return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	int rc;
	// 생략

	rc = im_add_strdata_to_complexdata("switch", "ON");
	if ( rc < 0 )	{
		printf("lastErrCode[%d]\n", im_get_LastErrCode());
		return -1;
	}
	// 생략
}
```

5.18 im_send_complexdata ()

복합데이터 관련 API 를 사용하여 구성된 수집데이터를 전송한다.

int im_send_complexdata(unsigned long long trxid)		
Param	trxid	이 데이터에 대한 트랜잭션아이디를 명시한다. 0값을 사용하면, sdk가 자동생성한다.
Return	성공/실패	= 0: 성공 < 0: 실패

코드예시)

```
#include <stdio.h>
#include <stdlib.h>
#include "iotmakers.h"

int main()
{
	int rc;

	// 생략
	rc = im_send_complexdata(0);
	if ( rc < 0 )	{
		printf("lastErrCode[%d]\n", im_get_LastErrCode());
		return -1;
	}
	// 생략
}
```

6 Sample 소스

6.1 수집데이터 전송하기

아래 소스코드는 IoTMakers 플랫폼에 접속 후에 1 초 간격으로 태그스트림 데이터를 전송하는 예제이다. 클라이언트 서비스를 종료하고 프로그램을 마치려면, 터미널에서 Ctrl-C(키보드 Ctrl 키와 문자키 'c'를 동시에)를 입력한다.

GatewayId, deviceId 그리고, devicePasswd 는 IoTMakers 포털에서 디바이스를 등록한 후에 디바이스 상세 정보 페이지에서 확인한다.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include "iotmakers.h"

static int local_loop = (0);

static void SigHandler(int sig)
{
	switch(sig)
	{
		case SIGTERM :
		case SIGINT :
			printf("accept signal SIGINT[%d]\n", sig);
			im_stop_service();
			local_loop = (0);
			break;
		default :
			;
	};
	return;
}
static void set_SigHandler()
{
	signal(SIGINT,   SigHandler);
	signal(SIGTERM,  SigHandler);
}

/* ============================
main_sample1.c

- Sending the collection data
	im_send_numdata();
	im_send_strdata();
============================ */
int main()
{
	int i;
	int rc;
```

```
	double val = 0.0;

	set_SigHandler();

	printf("im_init()\n");
	rc = im_init_with_config_file("./config.txt");
	if ( rc < 0  )	{
		printf("fail im_init()\n");
		return -1;
	}

	im_set_loglevel(LOG_LEVEL_DEBUG);

	printf("im_start_service()...\n");
	rc = im_start_service();
	if ( rc < 0  )	{
		printf("fail im_start_service()\n");
		im_release();
		return -1;
	}

	local_loop = (1);
	val = 0;
	while (local_loop == (1))
	{
		printf("im_send_numdata()...\n");
		rc = im_send_numdata("timer", val++, 0);
		if ( rc < 0  )	{
			printf("ErrCode[%d]\n", im_get_LastErrCode());
			break;
		}

		printf("im_send_strdata()...\n");
		rc = im_send_strdata("switch", "ON", 0);
		if ( rc < 0  )	{
			printf("ErrCode[%d]\n", im_get_LastErrCode());
			break;
		}
		sleep(5);
	}


	printf("im_stop_service()\n");
	im_stop_service();

	printf("im_release()\n");
	im_release();

	return 0;
}
```

6.2 수집데이터 전송(복합데이터)

아래 소스코드 예제는 2 개이상의 데이터를 동시에 전송하는 복합데이터 사용예제이다.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#include "iotmakers.h"

static int local_loop = (0);

static void SigHandler(int sig)
{
	switch(sig)
	{
		case SIGTERM :
		case SIGINT :
			printf("accept signal SIGINT[%d]\n", sig);
			im_stop_service();
			local_loop = (0);
			break;
		default :
			;
	};
	return;
}
static void set_SigHandler()
{
	signal(SIGINT,   SigHandler);
	signal(SIGTERM,  SigHandler);
}

/* ============================
main_sample2.c

- Sending the complex collection data
	im_init_complexdata();
	im_add_complex_data_number();
	im_send_complexdata();
============================ */
int main()
{
	int i;
	int rc;
	double val = 0.0;

	set_SigHandler();

	printf("im_init()\n");
	rc = im_init_with_config_file("./config.txt");
	if ( rc < 0  )	{
		printf("fail im_init()\n");
		return -1;
```

```
	}

	im_set_loglevel(LOG_LEVEL_DEBUG);

	printf("im_start_service()...\n");
	rc = im_start_service();
	if ( rc < 0  )	{
		printf("fail im_start_service()\n");
		im_release();
		return -1;
	}

	local_loop = (1);
	val = 0;
	while (local_loop == (1))
	{
		rc = im_init_complexdata();
		if ( rc < 0  )	{
			printf("ErrCode[%d]\n", im_get_LastErrCode());
			break;
		}

		rc = im_add_complex_data_number("Latitude", (double)
35.460670);
		if ( rc < 0  )	{
			printf("ErrCode[%d]\n", im_get_LastErrCode());
			break;
		}

		rc = im_add_complex_data_number("Longitude", (double)
126.914063);
		if ( rc < 0  )	{
			printf("ErrCode[%d]\n", im_get_LastErrCode());
			break;
		}

		printf("im_send_complexdata()...\n");
		rc = im_send_complexdata(0);
		if ( rc < 0  )	{
			printf("ErrCode[%d]\n", im_get_LastErrCode());
			return -1;
		}

		sleep(5);
	}


	printf("im_stop_service()\n");
	im_stop_service();

	printf("im_release()\n");
	im_release();

	return 0;
}
```

6.3 수집데이터 전송(트랜잭션아이디)

아래 소스코드는 사용자가 명시적으로 트랜잭션 아이디를 사용하고, 응답메시지에서 트랜잭션 아이디의 값을 확인하는 예제이다. 수집데이터 전송 후 응답 결과를 처리하기 위한 콜백함수를 등록했다.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#include "iotmakers.h"

static void mycb_dataresp_handler(long long trxid, char *respCode)
{
	printf("trxid=[%lld], respCode=[%s]\n", trxid, respCode);
}

static int local_loop = (0);

static void SigHandler(int sig)
{
	switch(sig)
	{
		case SIGTERM :
		case SIGINT :
			printf("accept signal SIGINT[%d]\n", sig);
			im_stop_service();
			local_loop = (0);
			break;
		default :
			;
	};
	return;
}
static void set_SigHandler()
{
	signal(SIGINT,   SigHandler);
	signal(SIGTERM,  SigHandler);
}

/* ============================
main_sample3.c

- Sending the collection data with transaction id
	im_set_dataresp_handler(mycb_dataresp_handler);

============================ */
int main()
{
	int i;
	int rc;
	double val = 0.0;
```

```
	set_SigHandler();

	printf("im_init()\n");
	rc = im_init_with_config_file("./config.txt");
	if ( rc < 0  )     {
		printf("fail im_init()\n");
		return -1;
	}

	im_set_loglevel(LOG_LEVEL_DEBUG);

	im_set_dataresp_handler(mycb_dataresp_handler);

	printf("im_start_service()...\n");
	rc = im_start_service();
	if ( rc < 0  )     {
		printf("fail im_start_service()\n");
		im_release();
		return -1;
	}

	local_loop = (1);
	val = 0;
	while (local_loop == (1))
	{
		long long trxid = im_get_new_trxid();
		printf("trxid=[%lld]\n", trxid);
		rc = im_send_numdata("timer", val++, trxid);
		if ( rc < 0  )     {
			printf("ErrCode[%d]\n", im_get_LastErrCode());
			break;
		}

		sleep(5);
	}


	printf("im_stop_service()\n");
	im_stop_service();

	printf("im_release()\n");
	im_release();

	return 0;
}
```

6.4 제어데이터 수신 및 처리

기본적으로 6.1 의 기능을 한다. 그리고, SDK 에서 내부 루프에서 수신한 제어데이터를 처리하기 위한 콜백함수를 등록했다.

디바이스에 제어데이터를 전송하려면, IoTMakers 포털에 로그인 한 후에 해당 디바이스의 상세 정보페이지에서, 제어타입의 태그스트림을 생성하고 제어데이터를 전송한다.
터미널에서 수신한 태그스트림의 이름과 값이 출력되는 것을 확인한다.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#include "iotmakers.h"

static int local_loop = (0);

static void SigHandler(int sig)
{
	switch(sig)
	{
		case SIGTERM :
		case SIGINT :
			printf("accept signal SIGINT[%d]\n", sig);
			im_stop_service();
			local_loop = (0);
			break;
		default :
			;
	};
	return;
}
static void set_SigHandler()
{
	signal(SIGINT,   SigHandler);
	signal(SIGTERM,  SigHandler);
}

static void mycb_numdata_handler(char *tagid, double numval)
{
	// !!! USER CODE HERE
	printf("tagid=[%s], val=[%f]\n", tagid, numval);
}
static void mycb_strdata_handler(char *tagid, char *strval)
{
	// !!! USER CODE HERE
	printf("tagid=[%s], val=[%s]\n", tagid, strval);
}

/* ============================
main_sample4.c
```

```
- Handling a control data
      im_set_numdata_handler(mycb_numdata_handler);
      im_set_strdata_handler(mycb_strdata_handler);
============================ */
int main()
{
      int i;
      int rc;

      set_SigHandler();

      printf("im_init()\n");
      rc = im_init_with_config_file("./config.txt");
      if ( rc < 0  )     {
            printf("fail im_init()\n");
            return -1;
      }

      im_set_loglevel(LOG_LEVEL_DEBUG);

      im_set_numdata_handler(mycb_numdata_handler);
      im_set_strdata_handler(mycb_strdata_handler);

      printf("im_start_service()...\n");
      rc = im_start_service();
      if ( rc < 0  )     {
            printf("fail im_start_service()\n");
            im_release();
            return -1;
      }

      local_loop = (1);
      while (local_loop == (1))
      {
            printf("waiting control data...\n");
            sleep(1);
      }

      printf("im_stop_service()\n");
      im_stop_service();

      printf("im_release()\n");
      im_release();

      return 0;
}
```

6.5 오류 처리

그리고, SDK 에서 발생한 오류코드를 처리하기 위한 콜백함수를 등록했다. 오류코드는 7.1 를 참조한다.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#include "iotmakers.h"

static int local_loop = (0);

static void SigHandler(int sig)
{
	switch(sig)
	{
		case SIGTERM :
		case SIGINT :
			printf("accept signal SIGINT[%d]\n", sig);
			im_stop_service();
			local_loop = (0);
			break;
		default :
			;
	};
	return;
}
static void set_SigHandler()
{
	signal(SIGINT,   SigHandler);
	signal(SIGTERM,  SigHandler);
}

static void mycb_error_handler(int lastErrCode)
{
	printf("lastErrCode=[%d]\n", lastErrCode);
	switch ( lastErrCode )
	{
	case 401:   //IM_ErrCode_SOCK_SEND_FAIL
	case 402:   //IM_ErrCode_SOCK_RECV_FAIL
	case 403:   //IM_ErrCode_SOCK_CONNECTION_FAIL
	case 404:   //IM_ErrCode_SOCK_CONNECTION_LOSS
		break;
	}
}



/* ============================
main_sample5.c

- Error event handler
```

```
	im_set_error_handler(mycb_error_handler);
============================ */
int main()
{
	int i;
	int rc;

	set_SigHandler();

	printf("im_init()\n");
	rc = im_init_with_config_file("./config.txt");
	if ( rc < 0  )	{
		printf("fail im_init()\n");
		return -1;
	}

	im_set_loglevel(LOG_LEVEL_DEBUG);

	im_set_error_handler(mycb_error_handler);

	printf("im_start_service()...\n");
	rc = im_start_service();
	if ( rc < 0  )	{
		printf("fail im_start_service()\n");
		im_release();
		return -1;
	}

	local_loop = (1);
	while (local_loop == (1))
	{
		printf("user looper...\n");

		printf("im_send_strdata()...\n");
		rc = im_send_strdata("switch", "ON", 0);
		if ( rc < 0  )	{
			printf("ErrCode[%d]\n", im_get_LastErrCode());
			break;
		}

		sleep(5);
	}

	printf("im_stop_service()\n");
	im_stop_service();

	printf("im_release()\n");
	im_release();

	return 0;
}
```

6.6 Makefile 샘플

```
##########################################
# This is a project standard makefile..
##########################################
SHELL   = /bin/sh
CC      = cc
AR      = ar ruvs

##########################################
# IOTMAKERS_SDK_HOME
##########################################
IOTMAKERS_SDK_HOME = ..
IOTMAKERS_SDK_LIBNAME = libiotmakers.a

IOTMAKERS_SDK_LIB_PATH = $(IOTMAKERS_SDK_HOME)/lib
IOTMAKERS_SDK_INC_PATH = $(IOTMAKERS_SDK_HOME)/include
IOTMAKERS_SDK_LIB = $(IOTMAKERS_SDK_LIB_PATH)/$(IOTMAKERS_SDK_LIBNAME)

##########################################
# SYSLIB
##########################################
# Linux
SYSLIB = -lpthread

##########################################
# FLAGs
##########################################
CFLAGS = -W -O0 -g
OFLAGS =
LIBALL = $(IOTMAKERS_SDK_LIB) $(SYSLIB)
INCLUDEALL = -I$(IOTMAKERS_SDK_INC_PATH)

##########################################
# SOURCE TREE
##########################################
C_OBJECT = \
        main.o

##########################################
# BUILD
##########################################
PRODUCT = tt
all: $(PRODUCT)

$(PRODUCT) : $(C_OBJECT)
        $(CC) -o $(PRODUCT) $(C_OBJECT) $(OFLAGS) $(LIBALL)

##########################################
# Compile
##########################################
.SUFFIXES : .o .asm .c .cxx .bas .sc .y .yxx .l .lxx  .pc

.c.o : $(C_SOURCE)
        $(CC) $(CFLAGS) $(INCLUDEALL) -c $*.c -o $*.o

.cxx.o : $(C_SOURCE)
        $(CC) $(CFLAGS) $(INCLUDEALL) -c $*.cxx -o $*.o

##########################################
# Util
##########################################
touch : $(C_SOURCE)
        rm -rf $(C_OBJECT)
```

7 부록

7.1 오류코드

SDK 내서 발생하는 발생하는 오류코드 목록이다. im_get_LastErrCode()함수가 오류코드를 반환한다.

코드	내용	비고
0	OK	에러 없음
100	UNKNOWN ERROR	정의되지 않은 오류
201	Config File Error	설정파일 오류
202	Memory Allocation Error	동적 메모리 할당 오류
301	Packet Head Parsing Error	헤더 파싱 실패
302	Packet Body Parsing Error	바디 파싱 실패
303	Packet Body Build Error	바디 생성 실패
304	Unknown Packet Received	알려지지 않은 패킷타입
401	Packet Send fail	패킷 전송 실패
402	Packet Receive fail	패킷 수신 실패
403	Connection fail	접속 실패
404	Connection loss	접속 소실

7.2 응답코드

수집데이터 전송에 대한 응답코드 목록이다.

코드	내용	비고
100	Success	처리성공
200	GeneralError	일반 오류
201	ImplementationError	구현 오류
202	PacketPushError	패킷푸쉬 오류
203	DecryptionError	복호화 오류
204	ParsingError	패킷파싱 오류
205	AuthenticationError	인증 오류
206	AckError	응답 오류
207	CommChAthnError	통신채널 인증오류
208	RequestInfoError	요청정보 오류

7.3 설정파일

SDK 초기화 시에 필요한 인자값을 설정파일을 통해서 불러올 수 있다. 각 항목의 이름과 의미는 다음과 같다.

이름	내용	예
SERVER_IP	IoTMakers 서버아이피	220.90.216.90
SERVER_PORT	IoTMakers 서버포트	10020
DEV_ID	디바이스 아이디	XXXXXXD1450836201263
DEV_PW	디바이스 패스워드	XXXXXX55o
GATEWAY_ID	Gateway연결ID	OPEN_TCP_001PTL001_1000999999
LOG_FILE_NAME	SDK로그파일명	/tmp/iotmakers_sdk.log